# Evaluation of Approximate Operators Case Study: Sobel Filter Application Executed on an Approximate RISC-V Platform

Geneviève Ndour, Tiago Trevisan Jost, Anca Molnos, Yves Durand, Arnaud Tisserand

# Evaluation of Approximate Operators Case Study: Sobel Filter Application Executed On An Approximate RISC-V Platform

Geneviève Ndour, Tiago Trevisan Jost, Anca Molnos, Yves Durand
CEA, LETI, Université Grenoble Alpes, Grenoble, France
{genevieve.ndour,tiago.trevisanjost,anca.molnos,yves.durand}@cea.fr

Arnaud Tisserand
CNRS, Labs-STICC UMR 6285, Université Bretagne Sud, Lorient, France
arnaud.tisserand@univ-ubs.fr

## ABSTRACT

Approximate computing techniques have been proposed to decrease the energy consumption or improve the performance of computation. Several designs of approximate arithmetic operators are proposed in the literature, suggesting that the operators' energy can be reduced up to 46%. This paper puts existing work in perspective by studying the impact of these operators in a processor targeting the embedded systems. We augment a RISC-V processor with approximate addition and multiplication that support a variable bitwidth in the range of 1 to 16 bits. We investigate the error-energy trade-off for a Sobel filter application. The results indicate that, for an acceptable output image degradation, 7.5% of the energy is saved. We conclude that the benefits of using approximate operators in embedded systems processors, have yet to be proved.

## KEYWORDS

Embedded systems, approximate computing, operators, energy

## 1 INTRODUCTION

Energy efficiency is a major concern in computing systems. Some emerging applications are error-tolerant, e.g., image processing, object recognition, data mining. Approximate computing is a new field that investigates the trade-offs between application output error and energy or performance [5]. In this context, many approximate operators are proposed. The evaluation of the individual operators indicates large gains in terms of energy consumption.

Embedded systems are frequently constraint in terms of energy. An existing way to exploit the approximate operators work in embedded systems is to integrate the operators themselves or their design principles within hardware accelerators [9]. In this paper we address a different question, namely the impact of approximate operators in an embedded system processor. For this an application-level evaluation is needed.

Existing application level evaluations of approximate operators are limited to the floating point number representation [1, 8]. However embedded systems often makes use of integer operators. Subsequently we augment a RISC-V processor [7] with approximate addition and multiplication and we investigate the errors due to approximations on a Sobel filter case study. The instruction breakdown of the Sobel filter indicates that the application contains 93.2% of non-approximable operations, such us memory accesses, and branches. We perform several optimizations to increase the number of approximable operations. The error-energy trade-off for the Sobel filter application, is performed on a bit range relevant for the image processing applications, i.e., 1 to 16 bits. The error metrics of interest for the Sobel filter are: the root mean square error (RMSE) [10], the peak signal to noise ratio (PSNR) [10] and the structural similarity (SSIM) [12]. Finally, we observe that, for an acceptable output image degradation, 0.95 SSIM, only 7.5% of the energy is saved.

The rest paper is organized as follows. Section 2 presents the related work, Section 3 describes the target architecture and the energy model, Section 4 presents the experimental results on the Sobel filter program and Section 5 concludes the paper.

## 2 RELATED WORK

Several approaches have been proposed in approximate computing at different layers to reduce the computation costs (e.g., energy). For example, instruction set architecture (ISA) extensions with approximate operations aim to reduce energy consumption [2]. In [1, 8] the authors propose to utilize 32-bit approximate floating point operators that allow the scaling of the operands bit-width. These studies present application-level error-energy trageoffs, however not for integer operators.

Several integer approximate operators have been also designed. In [6], Pagliari et al. propose a 16-bit multiplier using supply voltage scaling to reduce the bit-width of the operator. As a result energy consumption is reduced by up to 39%. In [3], Kahng et al. propose an accuracy-configurable approximate (ACA) adder which reduces energy consumption up to 30%. In [4], Kyaw et al. propose an error tolerant multiplier (ETM) that composed of two parts: a most significant bit (MSB) part for high accuracy and a least significant bits (LSB) part that allows certain level of error. The 12-bit ETM reduces energy consumption by up to 46%. We can observe that the energy savings in the above individual operators can achieve more than 30%. However, these operators have not been evaluated on a processor, at the application-level, to measure the global energy gains. The closed related work is the study of approximate floating point on the Sobel filter [1]. The results indicate energy gains with little impact in output quality. Nevertheless, this study is limited to precisions above 20 bits, and hence the most interesting bitwidths for image processing, i.e., under 16 bits, are not explored. To the best of our knowledge, we are the first to integrate approximate integer operator in a RISC-V processor and to evaluate their impact on an image processing application, namely a Sobel filter.


*SAMOS XVIII, July 2018, Samos Island-Grece*

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

## 3 TARGET ARCHITECTURE

The target architecture is a reduced instruction set computer (RISC) core, which is often used in low-power embedded systems. We employ the RISC-V 32-bit architecture template and supplement it with approximate operations. In what follows, the extension of RISC-V architecture and the energy consumption model are presented.

### 3.1 RISC-V architecture extended with approximate operations

For ease of explanation, we consider a classical 5-stage RISC-V pipeline. The pipeline stages are: *IF* (instruction fetch) to bring the instruction from the memory; *ID* (instruction decode) to decode the instruction; *EXE* (execution) to execute the instruction; *MEM* (memory) for data transfer instructions; *WB* (write back) to store the output data into the destination register. Note that our investigation is valid for other possible partitions of a RISC processor into pipeline stages.

We extend this RISC-V architecture by adding in the pipeline a unit to execute the approximate operators, denoted as *a.EXE*, as presented in Figure 1. *a.EXE* implements addition and multiplication with a configurable degree of approximation. In this work, approximation is implemented by executing the operation on a reduced bit-width, i.e., only on a configured number of most significant bits [6]. A separate instruction was added to configure the bit-width for *a.EXE*. All other RISC-V instructions are denoted as accurate instructions.

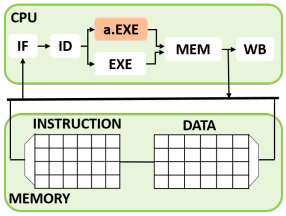


**Figure 1: RISC-V Architecture**

### 3.2 Energy model

To estimate the energy consumption of an application, we consider an energy model per instruction. This model is constructed starting with measurements obtained from a test-chip that includes a processor core with a similar pipeline as the RISC-V one and 256KB of memory, split into 64KB of instruction memory and 192KB of data memory. The energy is an average over multiple executions with random input data, and we assume that the energy of an instruction does not depend on the preceding instructions. In the rest of this paper, we present the energy values relative to multiplication. The energy ($e_{total}$) consumed by an instruction is the sum of the energy consumed by the instruction memory ($e_{MemInst}$), the energy consumed by the CPU ($e_{CPU}$) and the energy consumed by the data memory ($e_{MemData}$), as follows:

$$e_{total} = e_{MemInst} + e_{CPU} + e_{MemData} \quad (1)$$

To simplify the model, we split the RISC-V instructions into classes with similar energy consumption. For the accurate instructions the classes are: additions (add) that include additions and subtractions; multiplications (mul); logical instructions (logic), e.g., xor, and; branch instructions (br), e.g., jump, branch on greater than (bgt); store (st) and load (ld). The test chip allows separate measurements for the energy of the memory blocks and CPU. The total energy, $e_{total}$, is presented per class on Table 1. The approximate additions (a.add) and multiplications (a.mul) are considered to be each in a class of their own, to gain insights into their utilization. For these instructions, the $e_{CPU}$ depends on the bit-width ($bw$), as follows:

**Table 1: Relative energy values of the accurate instructions**

| Classes of instructions | relative energy values |
|---|---|
| add | 0.69 |
| mul | 1 |
| logic | 0.67 |
| br | 1.56 |
| st | 1.27 |
| ld | 0.9 |

$$e_{CPU}(bw) = e_{CPU^{base}} + e_{CPU^{a.EXE}}(bw), \quad (2)$$

where $e_{CPU^{base}}$ is the energy consumed on the stages IF, ID, MEM and WB in Figure 1, $e_{CPU^{a.EXE}}(bw)$ is the energy consumed to execute an approximate instruction. $e_{CPU^{a.EXE}}(bw)$ is obtained by fitting energy values from literature, in our case [6], such that $e_{CPU^{a.EXE}}(bw)$ is equal to 0 for 0 bit and for the maximum bit-width, $e_{CPU^{a.EXE}}(bw)$ is calculated with the following formula:

$$e_{CPU^{a.EXE}}(maxbw) = e_{CPU^{EXE}} \times (1 + f^{aovh}) \quad (3)$$

where $f^{aovh} > 0$ is the overhead energy of implementing the approximate operators due to the partitioning of the operators into various domains with different supply vector threshold, that introduces area overheads as we can see in [6]. In our case, $f^{aovh} = 0.1$. Figure 2 presents the relative energy values of the approximate instructions (a.add and a.mul), depending on the bit-width.

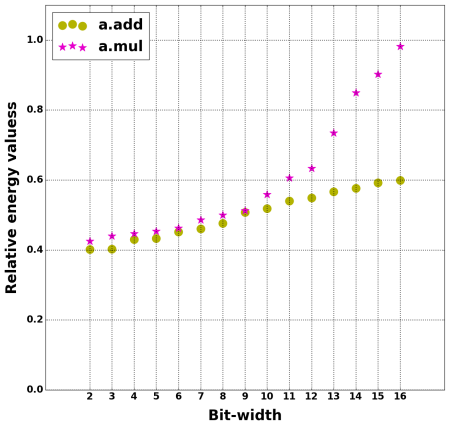


**Figure 2: Relative energy values of *a.add* and *a.mul***

## 4 EXPERIMENTAL CASE STUDY: THE SOBEL FILTER

The Sobel filter is a kernel used in image processing for edge detection. Edge detection is an image processing technique that identifies object boundaries in an image. The input is a RGB image and the

output is a gray-scale image in which the edges are emphasized. The image gradient of each pixel is calculated by convolving the image with a pair of filters (horizontal and vertical filters), that are 3x3 matrix.

To perform experiments, the RISC-V *spike* simulator [11] is augmented with the capabilities to count instructions and energy per class. The aim is to determine the fraction of the instructions that can be aproximated and investigate the error-energy tradeoff, as presented in the following.

## 4.1 Instructions breakdown

We utilize the Sobel filter source code provided by the Axbench benchmark [13]. However this implementation of the Sobel algorithm is in floating point and it is not optimised for the execution on a embedded processor. Consequently, we converted the source code from floating point to integer and we applied several optimizations typical for the low power domain. The optimisations are stacked-up one after the other, as follows:

- V1.: 2D vector for pixel storage is replaced by 1D vector to reduce the performance costs due to address calculations.
- V2.: V1 + the redundant code lines are removed to avoid to compute several times the same value, because the 3 components of a pixel have the same value in gray-scale image.
- V3.: V2 + the loops for convolution computations are unrolled to reduce the instructions introduced by the the loop indexing.
- V4.: V3 + remove filtering with null values. To apply the filters on the pixels that are at the border of the image, in the original version, the image is padded with zeros. In this optimisation we rewrite the filtering loops to remove this padding.

In Figure 3 we present the effect of these optimisations in terms of instructions breakdown. The percentage of approximate instructions is equal to 6.8% and 17% in the original version and the most optimized one, V4, respectively. Despite optimizations, most instructions are still not approximable (83% in the optimized one). These instructions consist of accurate arithmetic for address calculation, data memory access and branch instructions. The number of load instructions are generally large in comparison to the others. The high percentage of the load instructions is due to the fact that the pixels of the image are stored into the data memory and to compute the gradient of one pixel, the 3x3 matrix elements of the two filters and ones of the region of the pixel are loaded from the data memory.

In the next section, the most optimized version, V4, is utilized in the experiments.

## 4.2 Error-energy trade-off

To evaluate the errors induced to the Sobel filter by the reduce bit-width, we utilize three of the most common metrics proposed in image processing domain: the root mean square error (RMSE) [10], the peak signal to noise ratio (PSNR) [10] and the structural similarity (SSIM) [12]. The RMSE and the PSNR estimate the absolute errors between pixels of two compared images but are less correlated to human perception of image quality compared to the SSIM. To calculate the errors we consider as reference image the output of the original floating point Sobel filter. The experiments are performed on 100 input images.

Figure 4 presents the mean and the standard deviation of the RMSE and PSNR metrics with various bit-widths; the lower the RMSE, the higher the output quality is; the higher the PSNR, the higher the output quality is. The standard deviation (std) indicates that, for each bit-width, the metric value varies depending on the input image. The results indicate that from 8 bits we have an acceptable output image degradation.

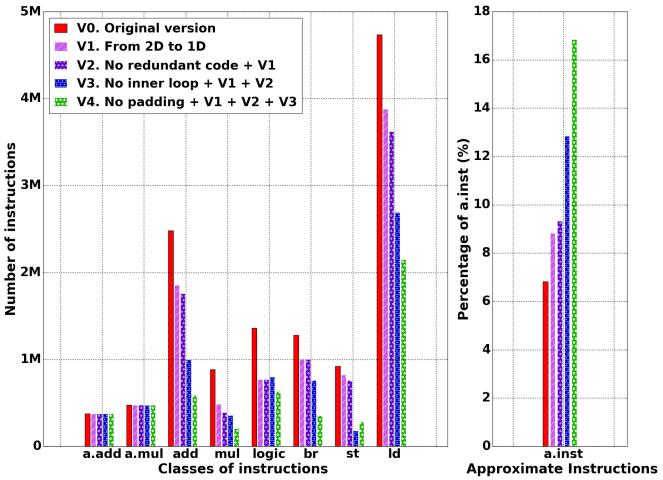


**Figure 3: Sobel filter instructions**

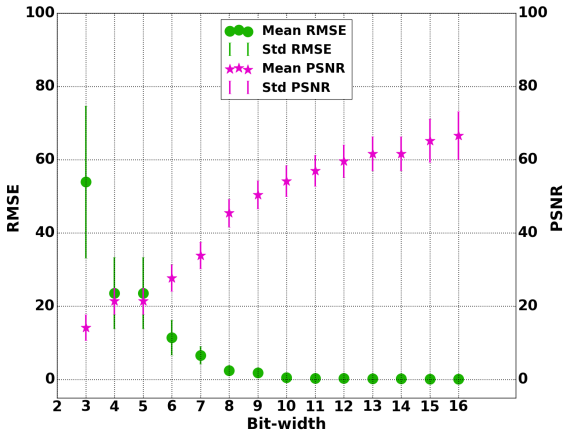


**Figure 4: RMSE and PSNR between the reference image and the approximate image**

Figure 5, evaluates the similarity between the reference and the approximate images. The results indicate that at least 8 bits are required to have an output image similar to the reference image.

Figure 6 shows that the energy gains vary according to the quality of the output and to have an acceptable output, the energy saved is equal to 7.5%.

The results indicate that, even after code optimization, the energy gains are not as large as the ones obtained evaluating the individual operators separately, as we can see in Table 2. The reduction of energy savings is due to the fact that the application includes more than 80% of non-approximate instructions.

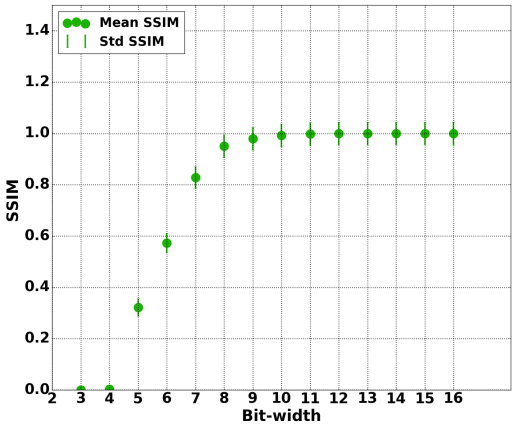


**Figure 5: SSIM between the reference image and the approximate image**

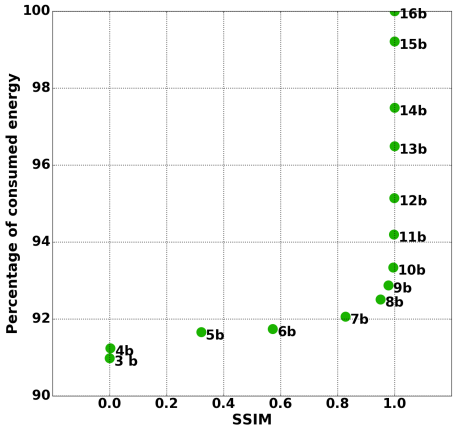


**Figure 6: Error-Energy trade-off**

**Table 2: Energy savings**

| Approximate operators/programs | Energy savings |
|---|---|
| Adder 16-bit [3] | 30% |
| Multiplier 12-bit [4] | 46% |
| Multiplier 16-bit [6] | 39% |
| Sobel filter | less than 10% |

## 5 CONCLUSION AND DISCUSSION

This paper presents an evaluation of common approximate operations, i.e., addition and multiplication, in a RISC-V processor, on a Sobel filter edge detection use-case. The operations are of integer type and the approximation therein is implemented by varying the bit-width of the operations. The Sobel output quality is evaluated with several metrics and the energy consumption is estimated. We employ an energy per instruction model derived from measurements on a test-chip.

The results indicate that even after optimizations, the fraction of the approximable instructions in the Sobel filter remains low, i.e., 17%. The memory accesses, the arithmetic instructions for computing addresses, and the branches, account for the largest fraction of the Sobel filter instructions and are not approximated. Furthermore, the Sobel filter, does not require more than 16 bits to execute with no errors when compared to a floating point implementation. This reduces the possible range of bit-width scaling, and hence of energy scaling. Note, this is the case for more image processing filters. Finally, the energy consumption of approximate addition and multiplications contains a part that does not scale down when the bit-width is reduced. These explain why the maximum energy savings are less than 10% for the Sobel filter.

To exploit approximate operations in embedded processors several directions for investigation are in sight. One is to extend the bit-width reduction also when accessing the data memory, i.e., load/store only the MSB bits. Note the other dominant instructions, i.e., branches, address computation, cannot be easily approximated without severe errors. Second, the study should be extended beyond 16 bits for embedded applications that require such a bit-width. Third, larger approximate operators, e.g., fused, have a higher chance of bringing energy gains without a significant impact in the quality of result.

## ACKNOWLEDGEMENTS

This work was partially supported by the French Agence Nationale de la Recherche, under Grant Agreement ANR-15-CE25-0015, project ARTEFACT. Furthermore, we would like to thank our colleagues from the Design and Architecture Lab for providing us with the energy values.

## REFERENCES

[1] Arun Chandrasekharan, Daniel Große, and Rolf Drechsler. 2017. ProACt: A Processor for High Performance On-demand Approximate Computing. In *Proceedings of the on Great Lakes Symposium on VLSI 2017*. ACM, 463–466.
[2] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. 2012. Architecture support for disciplined approximate programming. In *ACM SIGPLAN Notices*, Vol. 47. ACM, 301–312.
[3] Andrew B Kahng and Seokhyeong Kang. 2012. Accuracy-configurable adder for approximate arithmetic designs. In *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*. IEEE, 820–825.
[4] Khaing Yin Kyaw, Wang Ling Goh, and Kiat Seng Yeo. 2010. Low-power high-speed multiplier for error-tolerant application. In *Electron Devices and Solid-State Circuits (EDSSC), 2010 IEEE International Conference of*. IEEE, 1–4.
[5] Sparsh Mittal. 2016. A survey of techniques for approximate computing. *ACM Computing Surveys (CSUR)* 48, 4 (2016), 62.
[6] Daniele Jahier Pagliari, Yves Durand, David Coriat, Anca Molnos, Edith Beigne, Enrico Macii, and Massimo Poncino. 2017. A methodology for the design of dynamic accuracy operators by runtime back bias. In *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1165–1170.
[7] RISC-V 2018. (2018). https://riscv.org/.
[8] Adrian Sampson, Werner Dietl, Emily Fortuna, Danushen Gnanapragasam, Luis Ceze, and Dan Grossman. 2011. EnerJ: Approximate data types for safe and general low-power computation. In *ACM SIGPLAN Notices*, Vol. 46. ACM, 164–174.
[9] Jeremy Schlachter, Vincent Camus, and Christian Enz. 2016. Design of energy-efficient discrete cosine transform using pruned arithmetic circuits. In *Circuits and Systems (ISCAS), 2016 IEEE International Symposium on*. IEEE, 341–344.
[10] R Sivakumar and D Nedumaran. 2010. Comparative study of speckle noise reduction of ultrasound b-scan images in matrix laboratory environment. *International Journal of Computer Applications* 10, 9 (2010), 46–50.
[11] Spike simulator 2018. (2018). https://github.com/riscv/riscv-isa-sim.
[12] Zhou Wang, Alan C Bovik, and Hamid R Sheikh. 2005. Structural similarity based image quality assessment. *Digital Video image quality and perceptual coding* (2005), 225–241.
[13] Amir Yazdanbakhsh, Divya Mahajan, Pejman Lotfi-Kamran, and Hadi Esmaeilzadeh. 2016. AxBench: A Benchmark Suite for Approximate Computing Across the System Stack. (2016).